X10: The Big Picture
http://x10.sf.net
x10-users@lists.sourceforge.net

Vijay Saraswat
vijay@saraswat.org
July 2007
IBM Research

This material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0002.

Acknowledgments

- **X10 Core Team**
  - Rajkishore Barik, Ganesh Bikshandi, Chris Donawa, Allan Kielstra, Sreedhar Kodali, Nathaniel Nystrom, Igor Peshansky, Christoph von Praun, Vijay Saraswat, Vivek Sarkar, Lex Spoon, Pradeep Varma, Krishna Venkat, Tong Wen
- **X10 Tools**
  - Philippe Charles, Julian Dolby, Robert Fuhrer, Frank Tip, Mandana Vaziri
- **Emeritus**
  - Kemal Ebcioglu, Christian Grothoff, Vincent Cave
- **Research colleagues**
  - Ras Bodik, Guang Gao, Radha Jagadeesan, Jens Palsberg, Rodric Rabbah, Jan Vitek
  - Vinod Tipparaju, Jarek Nieplocha (PNNL)
  - Kathy Yelick, Dan Bonachea (Berkeley)
  - Doug Lea (SUNY Oswego)
  - Several others at IBM

Recent Publications

1. “Constrained Types for OO Languages”, Submitted.
2. “Deadlock-free scheduling of X10 Computations with bounded resources”, SPAA 2007.
3. “A Theory of Memory Models”, PPoPP 2007.
4. “May-Happen-in-Parallel Analysis of X10 Programs”, PPoPP 2007.
5. “An annotation and compiler plug-in system for X10”, IBM Technical Report, Feb 2007.
6. “Experiences with an SMP Implementation for X10 based on the Java Concurrency Utilities” Workshop on Programming Models for Ubiquitous Parallelism (PMUP), September 2006.
7. "An Experiment in Measuring the Productivity of Three Parallel Programming Languages”, P-PHEC workshop, February 2006.
8. "X10: An Object-Oriented Approach to Non-Uniform Cluster Computing", OOPSLA conference, October 2005.
9. "Concurrent Clustered Programming", CONCUR conference, August 2005.
10. "X10: an Experimental Language for High Productivity Programming of Scalable Systems", P-PHEC workshop, February 2005.

Tutorials

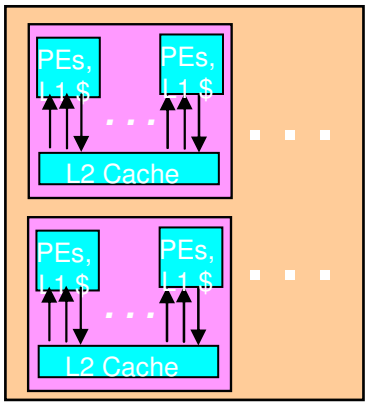
- TiC 2006, PACT 2006, OOPSLA 2006, PPoPP 2007
- Graduate course on X10 at U Pisa (07/07)

A new era of mainstream parallel processing

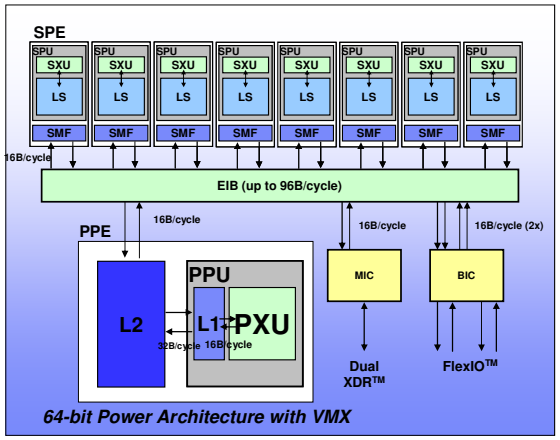
The Challenge

Parallelism scaling replaces frequency scaling as foundation for increased performance ➔ Profound impact on future software

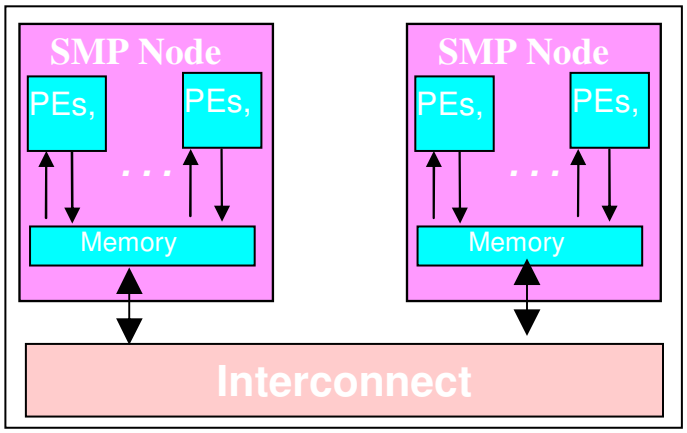
Multi-core chips



Heterogeneous Parallelism



Cluster Parallelism



Our response:

Use X10 as a new language for parallel hardware that builds on existing tools, compilers, runtimes, virtual machines and libraries

A strategy – radical incrementalism

- **One (mainstream) programming model to rule them all.**
  - Multicore, Clusters, Combos
  - Sequential Java + concurrency
  - places, async, finish, clock, atomic, annotations
- **Keep it simple, stupid**
  - Look to aggressively parallelize sequential programs
  - Concurrency as a means to an end
- **Address all approaches**
  - VM extensions
  - Libraries
  - Annotations + extensive static analysis
  - New languages
  - Domain-specific parallelism

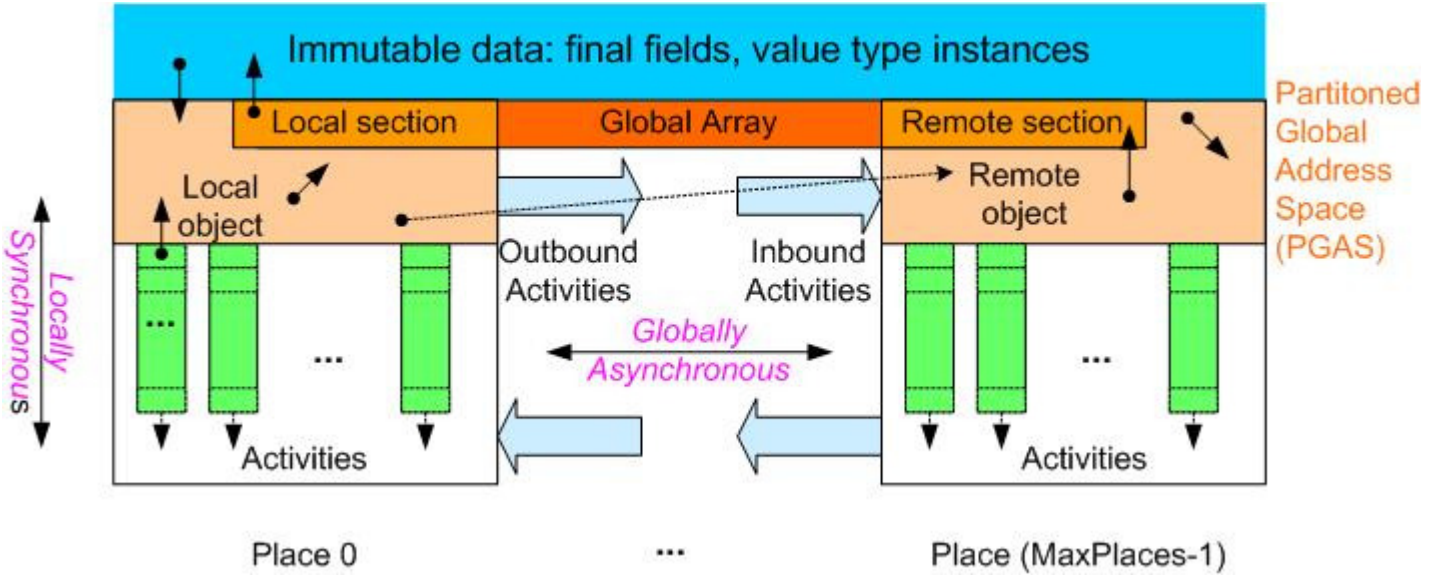
Themes

- **Target productivity, without sacrificing performance.**
- **Build on what we understand – core sequential OO base.**
- **Manage concurrency and distribution explicitly.**
- **Target the "mainstream" programmer; do not ignore the expert.**
- **Rule out large classes of errors by design – statically as far as possible, else dynamically.**
- **Keep the language small, orthogonal.**
- **Make easy things easy, hard things possible.**
- **Address the entire tool chain: development environment, analysis tools, debugging, …**

Programming Model -- The Big Picture

- **Places**
  - Partition data and activities that operate on the data
  - But keep address space global.
  - *Locality central to heterogeneity, clustering.*
- **Asynchrony**
  - Express fine-grained parallelism
  - Exploit fork-join structure, recursive partitioning.
- **Atomicity**
  - Specify granularity of atomicity.
  - *Focus on the desired property, not the mechanism*.
- **Ordering**
  - Termination detection
  - Quiescence detection

X10 Programming Model – The Big Picture



Storage classes:

- **Activity-local**
- **Place-local**
- **Partitioned global**
- **Immutable**

- Dynamic parallelism with a *Partitioned Global Address Space*
- *Places* encapsulate binding of activities and globally addressable data
  - Number of places currently fixed at launch time
- All concurrency is expressed as *asynchronous activities* – subsumes threads, structured parallelism, messaging, DMA transfers, etc.
- *Atomic blocks* enforce mutual exclusion of co-located data
  - No place-remote accesses permitted in atomic section
- *Immutable* data offers opportunity for single-assignment parallelism

X10 v1.01 Cheat sheet

Stm:

async [(*Place*)] [clocked *ClockList*] *Stm*

atomic *Stm*

when (*SimpleExpr*) *Stm*

finish *Stm*

next; *c.resume()* *c.drop()*

for(*i : Region*) *Stm*

foreach (*i : Region*) *Stm*

ateach (*I : Distribution*) *Stm*

Expr:

ArrayExpr

ClassModifier : Kind

MethodModifier:

atomic nonblocking sequential

local safe

Type:

DataType

nullable<*Type*>

future <*Type* >

Kind :

value | reference

ClassType , InterfaceType :

TypeName

[DepParameters] [PlaceTypeSpecifier]

Annotations: @ InterfaceType

Annotations suffix types and prefix almost all other syntactic elements.

x10.lang has the following classes (among others)

point, range, region, distribution, clock, array

Some of these are supported by special syntax.

X10 v1.01 Cheat sheet: Array support

ArrayExpr:

- new ArrayType (Formal) { Stm }
- *Distribution Expr* -- Lifting
- *ArrayExpr* [*Region*] -- Section
- *ArrayExpr* | *Distribution* -- Restriction
- *ArrayExpr* || *ArrayExpr* -- Union
- *ArrayExpr*.overlay(*ArrayExpr*) -- Update
- *ArrayExpr*. scan(*[fun [, ArgList]*)
- *ArrayExpr*. reduce(*[fun [, ArgList]*)
- *ArrayExpr*.lift(*[fun [, ArgList]*)

ArrayType:

- *Type [Kind]* [ ]
- *Type [Kind]* [region(N)]
- *Type [Kind]* [*Region*]
- *Type [Kind]* [*Distribution*]

Region:

- *Expr : Expr* -- 1-D region
- [*Range, …, Range*] -- Multidimensional Region
- *Region* && *Region* -- Intersection
- *Region* || *Region* -- Union
- *Region* – *Region* -- Set difference
- *BuiltinRegion*

Dist:

- *Region -> Place* -- Constant distribution
- *Distribution* | *Place* -- Restriction
- *Distribution* | *Region* -- Restriction
- *Distribution* || *Distribution* -- Union
- *Distribution* – *Distribution* -- Set difference
- *Distribution*.overlay (*Distribution*)
- *BuiltinDistribution*

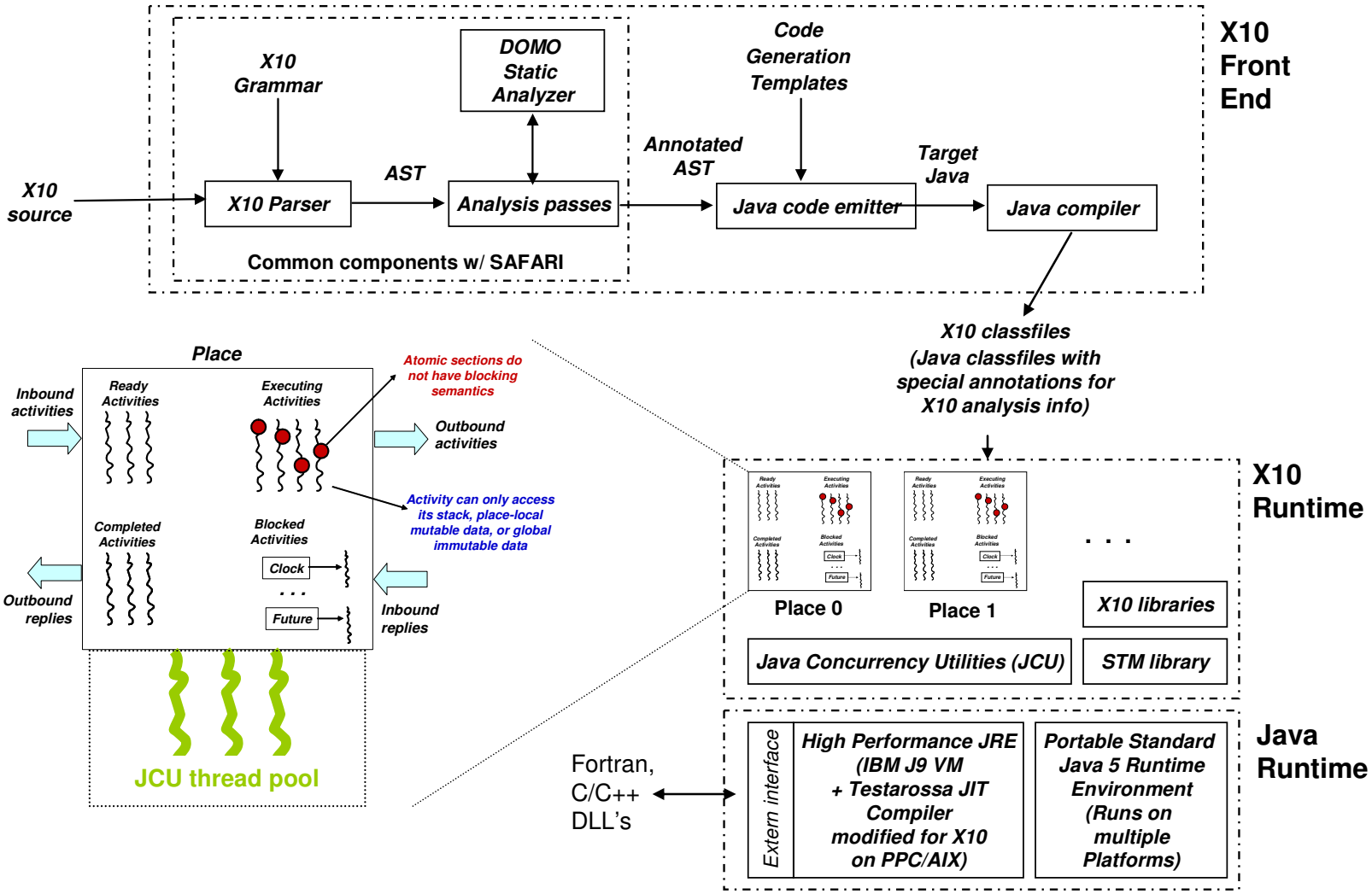
Language supports type safety, memory safety, place safety, clock safety.

IBM Research

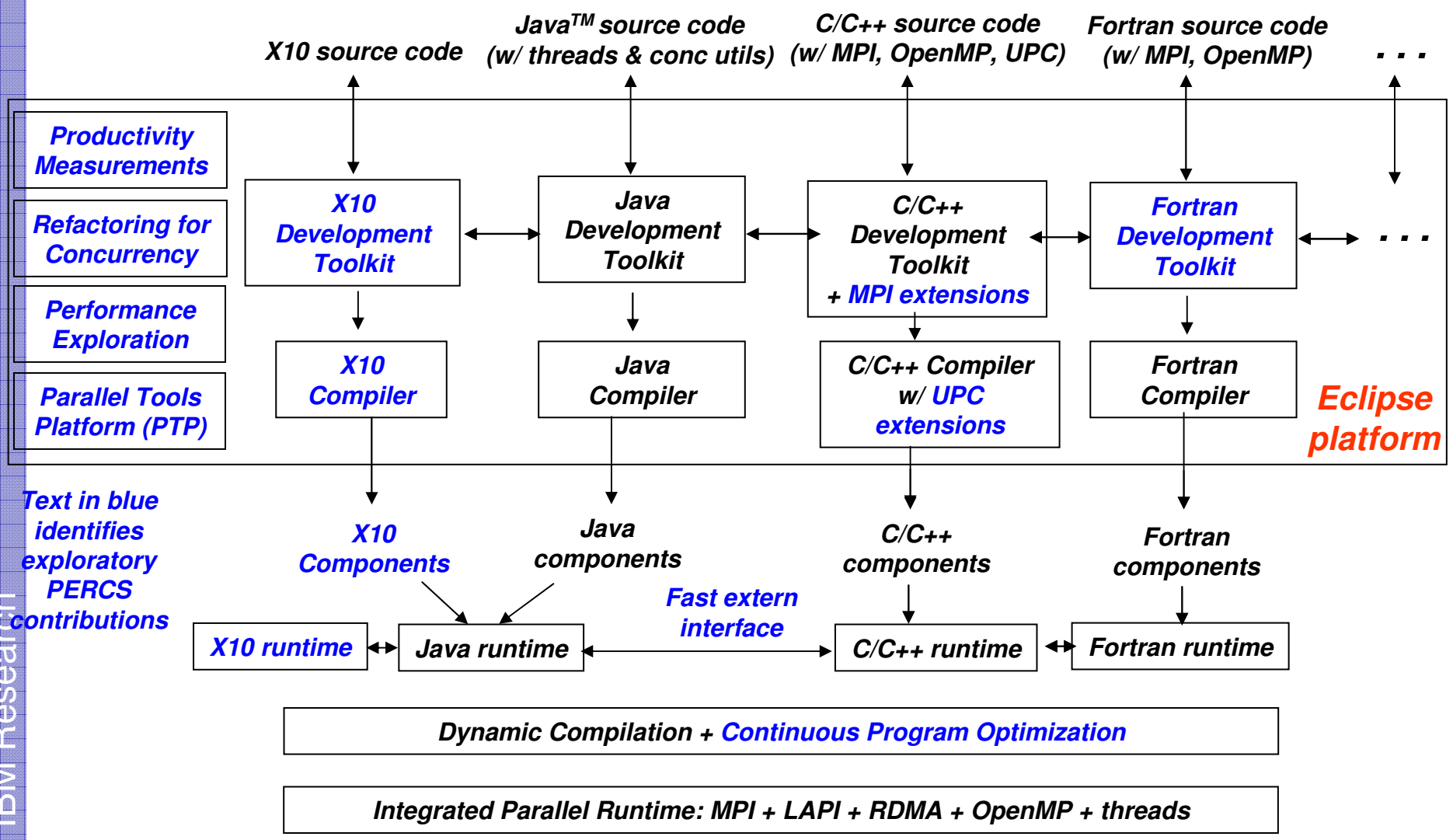
X10 availability

- **X10 is an open source project (Eclipse Public License).**
- **Website: http://x10.sf.net**
- **Reference implementation in Java, runs on any Java 5 VM.**
  - Windows/Intel, Linux/Intel
  - AIX/PPC, Linux/PPC
  - Runs on multiprocessors
- **Website contains**
  - Tutorial material
  - Presentations
  - Download instructions
  - Copies of some papers
  - Pointers to mailing list

Multi-core SMP Implementation



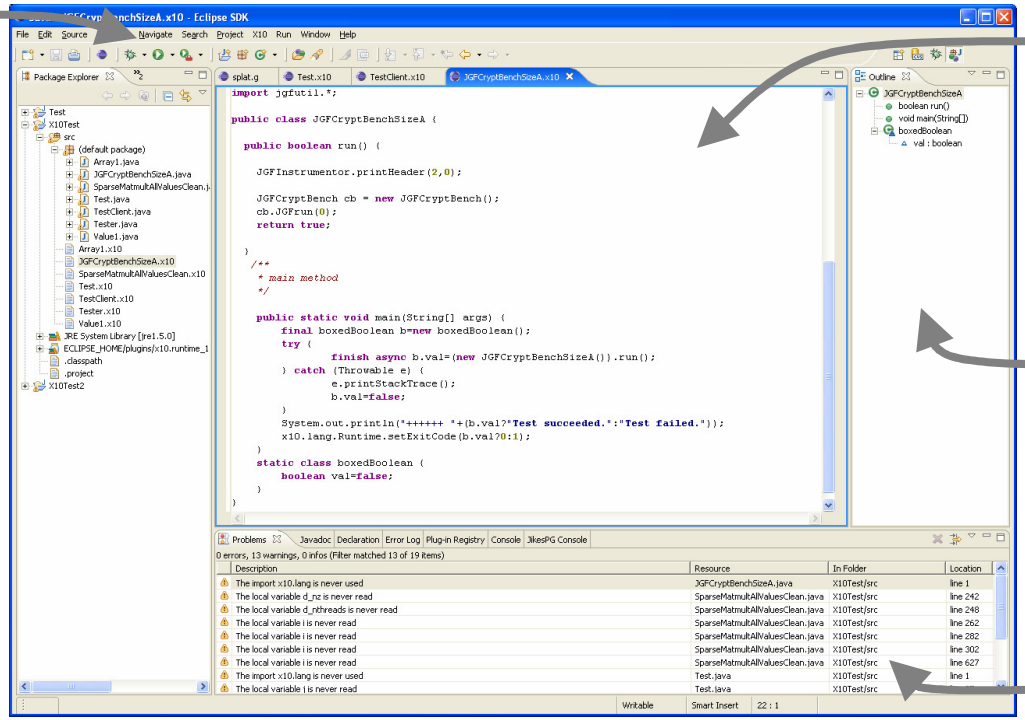
PERCS Programming Model, Tools



IBM Research

X10DT: Enhancing productivity

X10 Launch Configuration



Source editor w/ syntax highlighting, auto indenting, some content assist

Outline View populated w/ X10 types, members, loops

X10 Incremental Builder; Problems View populated w/ X10 compiler messages

- **Code editing**
- **Refactoring**
- **Code visualization**
- **Data visualization**
- **Debugging**
- **Static performance analysis**

Vision: State-of-the-art IDE for a modern OO language for HPC

X10 Compiler (06/2007)

09/03 PERCS Kickoff

02/04 X10 Kickoff

07/04 X10 0.32 Spec Draft

02/05 X10 Prototype #1

07/05 X10 Productivity Study

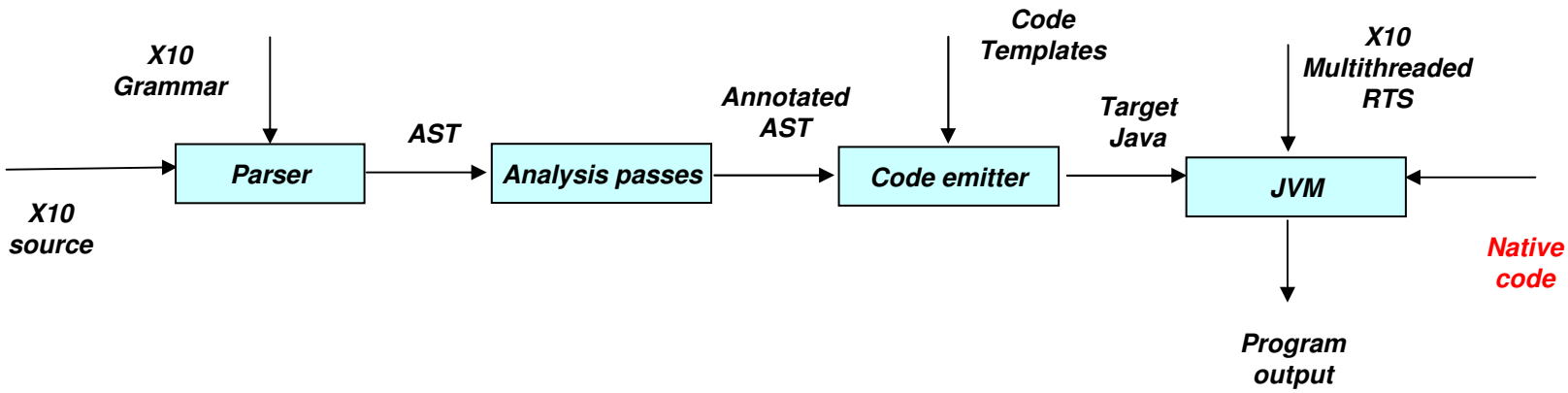
12/05 X10 Prototype #2

12/06 Open Source Release

6/07 Annotations, X10lib v1

IBM Research

Operational X10 implementation (since 02/2005)



Structure

- Translator based on Polyglot (Java compiler framework)
- X10 extensions are modular.
- Uses Jikes parser generator.

Code metrics

- Parser: ~45/14K*
- Translator: ~112/9K
- RTS: ~190/10K – revised for JUC
- Polyglot base: ~517/80K
- Approx 280 test cases.

(* classes+interfaces/LOC)

New features 6/07

- Annotations
- X10lib v 1

X10Flash

- **Distributed runtime**
  - In C/C++
  - On top of messaging library (GASNet, ARMCI, LAPI)
  - Targeted for high-performance clusters of SMPs.
- **X10lib**
  - Runtime also made available as a standalone library.
  - Supporting global address space, places, asyncs, clocks, futures etc.
- **Performance goal**
  - To be competitive with MPI
- **Release schedule**
  - Internal demonstration 12/07
  - External release 2008

Conclusion

- **X10 is intended to span multicore, heterogenous systems and clusters.**
- **X10 offers a simple programming model for concurrency and distribution**
  - Places
  - Asynchrony
  - Atomicity
  - Ordering
- **X10 is being developed as an open source project**
  - Reference implementation available on http://x10.sf.net
- **A clustered implementation is being worked on**
  - Compiles to C/C++
  - Uses LAPI for inter-process messaging
  - Uses algorithmic scheduling for intra-process scheduling

Backup

Comparison with MPI

Model of parallelism

- MPI: data parallel, SPMD
- X10: structured multithreading, SPMD is a special case.

Synchronization model

- MPI: clear distinction between control synchronization (barriers) and communication (or combine them in a collective).
  - synchronous: send / receive
  - asynchronous: put/get
- X10: shared memory paradigm
  - address space partitioned,
    - locality of access visible to the programmer
  - remote access are asynchronous (but may be made synchronous)
  - Supports “active messages”.

Comparison with UPC

X10 is similar to UPC in ...

- shared partitioned global address space
- barrier synchronization:
  - UPC split barrier (upc_notify, upc_wait)
  - X10 clocks: resume(), next
- parallel loops with affinity
  - UPC affinity clause in upc_forall
  - X10 distribution in ateach

X10 extends UPC in ...

- dynamic structured multithreading (not SPMD)
- safety properties (managed runtime)
- distributed computation and data, concept of places
- array language, multidimensional arrays
- critical section synchronization
  - simple atomic blocks, not locks
  - conditional atomic blocks
- type system exposes access locality
- multiple, custom distributions

Comparison with CILK

X10 is similar to CILK in ...

- structured concurrency
  - Cilk: spawn, sync
  - X10: async, finish (block scoped, rooted exception model), method body not forced to finish spawned asyncs.
  - serial elision only for non-blocking activities in X10

X10 extends CILK in ...

- distributed computation and data, concept of places
  - intra/inter-place work-stealing scheduler?
- array language, multidimensional arrays
- critical sections
  - simple atomic blocks, not locks
  - X10 has condition synchronization
- X10 managed runtime and type system: safety properties

Comparison with Java ™

X10 language builds on the Java language

Shared underlying philosophy: shared syntactic and semantic tradition, simple, small, easy to use, efficiently implementable, machine independent

X10 does not have:

- Dynamic class loading
- Java's concurrency features
  - thread library, volatile, synchronized, wait, notify

X10 restricts:

- Class variables and static initialization

X10 adds to Java:

- **value types, nullable**
- **Array language**
  - Multi-dimensional arrays, aggregate operations
- **New concurrency features**
  - activities (async, future), atomic blocks, clocks
- **Distribution**
  - places
  - distributed arrays

Overview of green field work

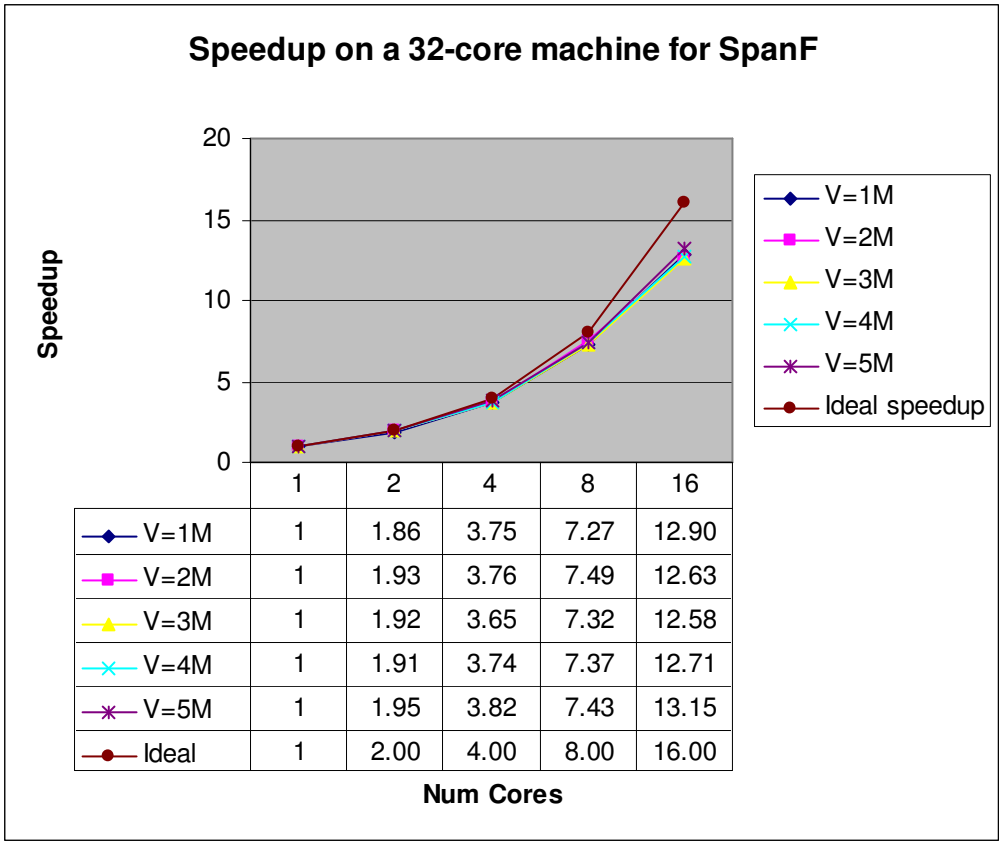
- **Compilation for Combos**
  - Cell + Opteron
  - Graphics cards – NVidia (CUDA)
- **Implement specific annotations and supporting static analyses**
  - Integrate with WALA (http://wala.sf.net)
- **Support Implicit Parallelism**
  - tryasync, dependency annotations, flow annotations, …
  - Static/dynamic support
- **Runtime**
  - x10lib – an implementation of the X10 runtime in C/C++ for high-performance clusters
  - Algorithmic scheduling
    - Design extensions to handle X10 control constructs.
    - Evaluate different scheduling strategies (work-stealing, parallel depth-first scheduling, …)
    - Implement

Algorithmic Scheduling

- **Centralized task-queue is a bottleneck**
  - Each worker must synchronize in order to get new work
- **Thread pool may grow unboundedly**
  - when causes thread to suspend.
- **Observation: finish/async programs have series/parallel dependence graphs.**
- **Solution: use Cilk-style work-stealing scheduler**
  - Guaranteed $O(T_1/p + T_\infty)$ runtime, and $p*S_1$ space.
- **Todo:**
  - Investigate alternate schedulers with better space utilization/cache behavior (cf parallel depth-first scheduler)
  - Consider hiearchical, multi-place scheduler
  - Handle all X10 control constructs.

Preliminary results: very encouraging. Good scaling, absolute performance.

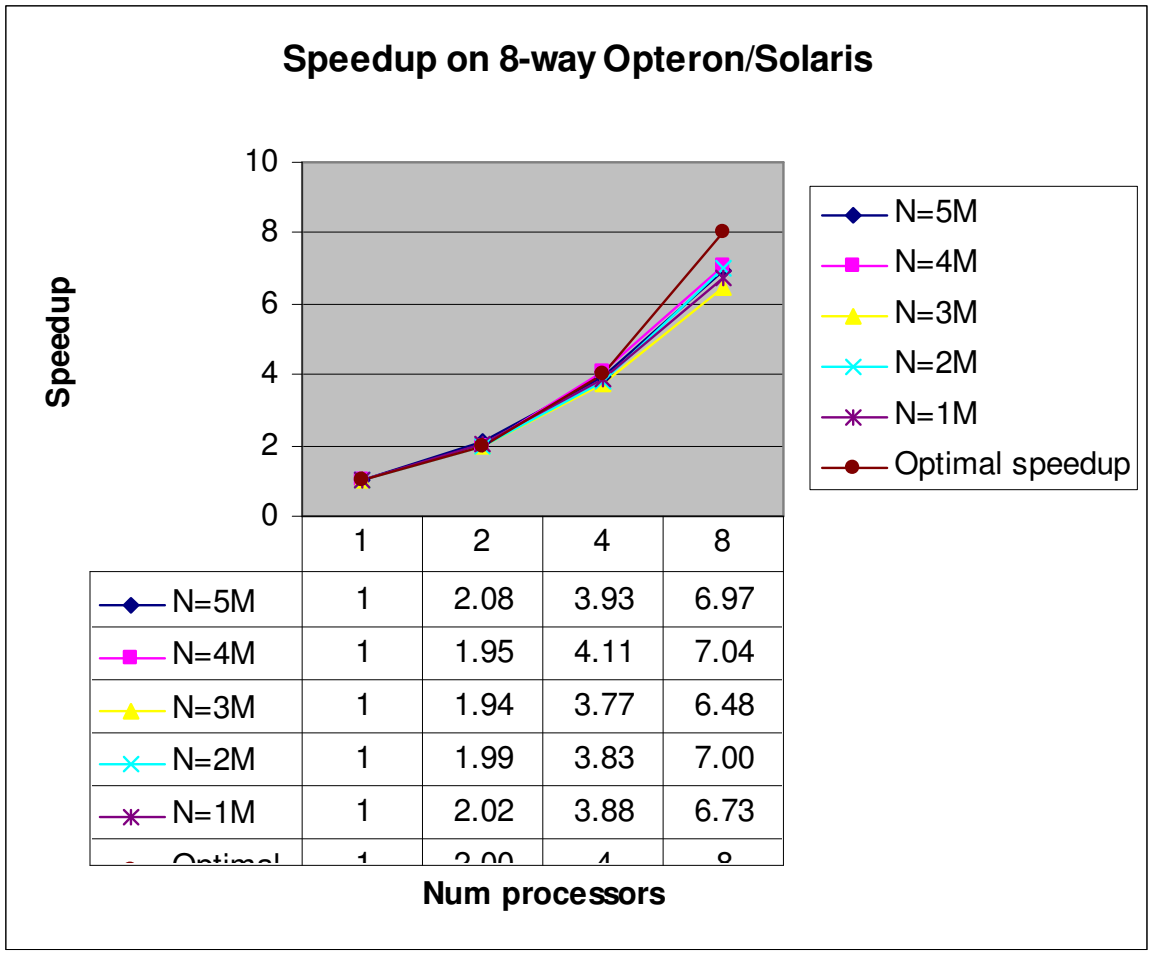
Speedup for spannning tree on a 32-core Niagara



	1	2	4	8	16
V=1M	1	1.86	3.75	7.27	12.90
V=2M	1	1.93	3.76	7.49	12.63
V=3M	1	1.92	3.65	7.32	12.58
V=4M	1	1.91	3.74	7.37	12.71
V=5M	1	1.95	3.82	7.43	13.15
Ideal	1	2.00	4.00	8.00	16.00

Runtime ~ 0.5s for p=24, V=5M, E=20M

Speedup for spanning tree on 8-proc Opteron/Solaris



	1	2	4	8
N=5M	1	2.08	3.93	6.97
N=4M	1	1.95	4.11	7.04
N=3M	1	1.94	3.77	6.48
N=2M	1	1.99	3.83	7.00
N=1M	1	2.02	3.88	6.73
Optimal	1	2.00	4	8

Runtime ~ 1.19s for p=8, V=5M, E=20M